UiO : Department of Informatics
University of Oslo

Machine learing in monitoring systems

How can machine learning be used to understand large quantities of data from monitoring.

Gisle Bøhn Hagen

Master's Thesis Spring 2014

Machine learing in monitoring systems

Gisle Bøhn Hagen

20th May 2014

Abstract

This thesis addresses the problem of monitoring for system administrators. Can monitoring be done better with a mathematical algorithm to improve the ratio between information vs noise and gain a overall better out put from the monitoring? By solving these problems system administrators will gain a better situational awareness of their systems and therefor can work more efficiently. This is much needed, because as systems increase in scale, the responsibilities and complexity for system administrators also increases. The work in the thesis will look into how this can be done with different prototypes and experiments, using Sensu monitoring and the EventRank algorithm. Sensu utilises the monitoring-as-code principle and is therefore ideal to combine with EventRank. Finally, the results will later be analysed in order to measure the stressfulness of the experiment, and will be used to consider wether the problem statement has been answered.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to extend my gratitude to my supervisor, Kyrre Begnum. Thank you for your guidance and effort that has helped me complete my master thesis. I would also like to thank my friends and family who have supported me, with a special thanks my roommates Anette and Synne for improving my writing skills. Finally, i would like to thank my fellow students who have helped with discussions, technical support and motivation during low periods.

Chapter 1

Introduction

System administration is a large part of the computer science world and has many difficult and demanding tasks. A system administrator is responsible for large and very complex systems, and his tasks vary every day - from installing new hardware into a cloud environment to automating a process with a script. System administrators today have to handle systems consisting of up to a thousand different customers - each of who possess large amounts of hardware, from desktop computers and servers to routers and switches. The complexity created by the continuously growing systems, creates risk. Calculating risk is an important part of the system administrator's job as they are responsible for up time of all the systems. An example is Altinn[1], the Norwegian online tax reviewing system, that malfunctions every year when tax returns are published. This is a classic example as it only malfunctions once a year when the load intensifies, but works normally for the rest of the year when the load is normal. Here the system administrator has to make an assessment of the risk this involves, and make a decision if the system needs changing.

As mentioned, a large part of being a system administrator is making sure every part of the system is up and running at all times. This requires monitoring, a task all system administrators need to do in their daily operations. Monitoring provides a deeper insight into all the components in a system architecture. Creating monitoring setups for large systems is a complex process, as it needs to be installed on every component where monitoring is required. Just one company can have up to a million different devices that need monitoring, and this generates great quantities of data. This is one of the reasons monitoring can be so difficult to implement and configure, because if it is done wrong it will generate false positives and extra work. Larger enterprises have people working full time with monitoring to assure up time for companies. They immediately respond if something malfunctions. If a part of the system goes down, for example the network connection, it can effect the business value, as can be seen on Figure 1.1.

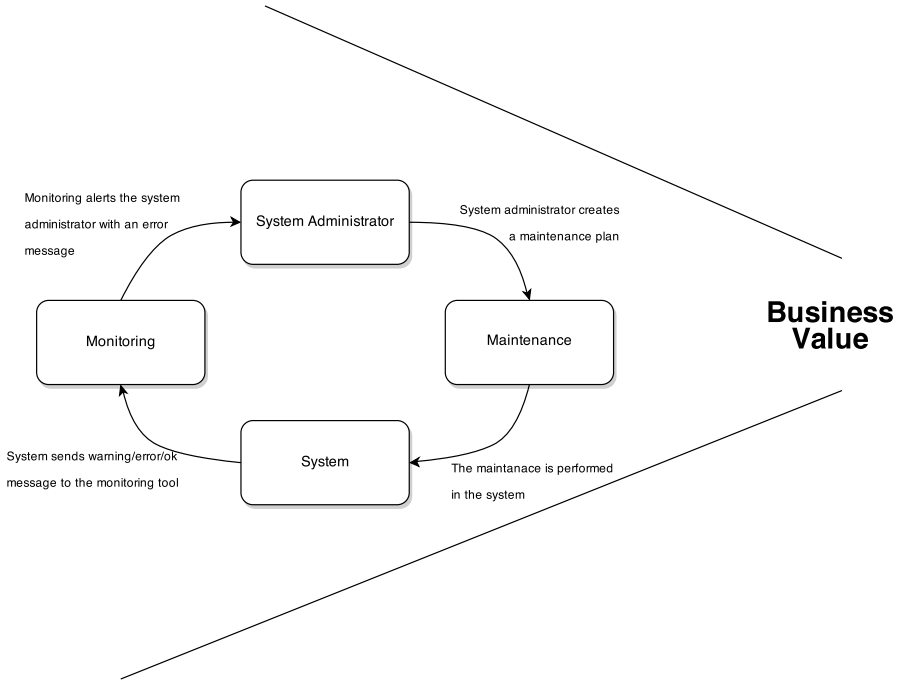

Figure 1.1: Concept of monitoring compared to Business value

Monitoring generally means to be aware of the state of a system. Not only does it alert when there is a malfunction, it also gives insight if there is a warning state on a component, or showing that everything is functioning normally. Without monitoring it is almost impossible for a system administrator to know in which state the system is currently operating. At the moment there are many tools available on the market that offer monitoring for instance Sensu, a monitoring tool for cloud environments. Monitoring tools like Sensu[2] utilises passive monitoring, where a control centre watches the traffic sent by the nodes. This type of monitoring requires that the nodes are capable of transmitting network data with the status at all times, so the control centre can analyse the input it receives. This type of monitoring needs implementation work on every node, but little maintenance in the long run as they are self sustained.

It is common to use monitoring tools to help with monitoring. The tools help gather information through event monitoring and alert if an anomaly occurs in the system. These alerts are triggered when an event is detected, and can be anything from a disk failure to a network not responding. Tools like this detect errors by use of a logical bus between the sender and reciver and respond if the answer is incorrect or unresponsive. This means that if the sender is not able to respond, the system administrator will get an alert of a malfunction in the system by the monitoring server. By using this type of tool with event monitoring, it is much easier to control the system and the events it generates.

While monitoring can give the system administrator large amounts of useful information, it can also be complicated to configure and manage. If a monitoring system is not configured properly, it can generate many false positives and alert the system administrator incorrectly. Another problem is that when many events happen at the same time, distinguishing out the important one can be difficult and time consuming. All of this can lead to inaccurate monitoring, which can be devastating for the operations of a system administrator, who then would use most of his time fire distinguishing instead of continuing development of the system.

The events in monitoring can be described as an action or occurrence that can be detected by the monitoring system. These events can be many, and it can be difficult to sort out the important one from the false positives. When trying to sort the event it can be hard to see patterns in the anomalies which can help detect and solve problems in the system. If events can be sorted in a better way, it can help finding problems much faster, and therefore save time by seeing the bigger picture of the system.

A common way to attack problems with false positives is by using algorithms of fluctuating advanced form to either distill, compress or highlight interesting qualities which is not part of the information itself. Highlighting interesting qualities, often called anomaly detection will try to detect statistical qualities of events and to this way discover errors in the system. For example a message that normally would be interpreted as routine, would surface in an irregular time area. In essence, monitoring faces a problem with information against noise. If we look at the flow of messages we get from monitoring systems as a signal, the challenge will always be to extract information from the signal and simultaneously eliminate the noise.

Traditionally people have been responsible for this filtering as state of the art systems only offers policy based filtering for example, critical is more important than warning and filtering on customer names. It stands to reason that through the explosive increase of systems and architectures that our problem with information vs noise have become a problem of scale. To be able to ease the load on humans we need to find new ways to extract information from signals so that the focus can be directed to more places where it is more interesting and important. A new trend in monitoring is to design systems that facilitate scale and dynamics, and create systems that focus on adding pieces of code. One example of this is Sensu, as mentioned before. A big part of Sensu functionality is decided not by configuration files, but by adding codes in elements. This means that new systems go away from large static configurations and into monitoring-as-code instead. An example of this is illustrated in figure 1.2.

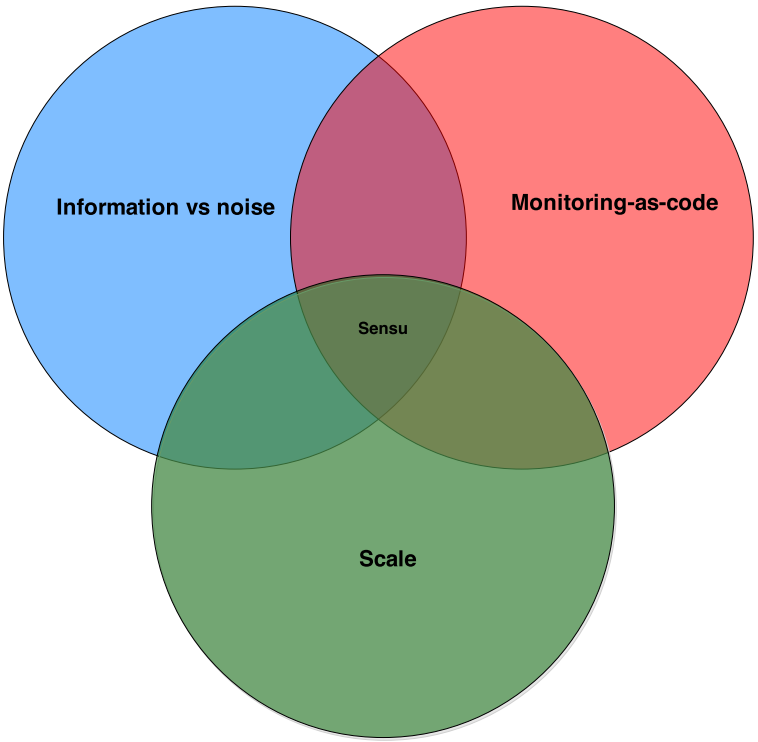

Figure 1.2: A complete monitoring system

To be able to introduce filtering and compression of information we need to add code to monitoring-as-code systems, and implement algorithms from the information vs noise principle. This needs to be done so we can get the best from both worlds and create a better monitoring system. A method that can sort events would be the EventRank algorithm that tries to identify patterns in how events behave together. This can give an interpretation of information given that events happening most often together can be important and have a high information value. Conversely one can say that events happening at the same time, but has never done so before also might have an information value. This leaves us with the practical problem:

Practical problem: How can mathematical algorithms be used to understand large quantities of data from monitoring.

By solving this problem it is possible to save large amounts of time troubleshooting for problems based on events. This means that it is easier to find the source of the problem from the group of event that occurred at the same time. It will also reduce the complexity of the output from monitoring tools by combining the event and therefor increase the situational awareness. This is important for making the right decisions and choices when managing large systems where errors can have huge consequences and be very expensive.

By reducing the output from event monitoring, it reduces the overall management. This way the system administrator can focus more of his energy into development and tuning the system, rather than just maintenance and fire distinguishing. This leaves us with the final problem statement:

1.1 Problem Statement

Problem statement How can EventRank be used to improve situational awareness of events using a monitoring system

This thesis will look at which advantages a monitoring system will gain by implementing EventRank in addition to normal operations. It will explore how the system can process incoming events and gain a better understanding by the output. The algorithm will be implemented in Sensu monitoring and run through several experiments to see if the output improves. When this is said, the goal is not to create a complete monitoring system, but rather to look at the improvements possible to make.

Chapter 2

Background

This capter will look at important principles and technologies used in this thesis. It will take a closer look at monitoring-as-code, Sensu, information vs noise and the EventRank algorithm. These are important principles that needs researching before starting the development. This is so that the base of the thesis is well founded and the work can be correctly done.

2.1 Monitoring-as-code

Today monitoring is widely used and needed for operational purposes. As mentioned earlier monitoring is traditionally done with with static configuration files that determine what values to monitor. This is a good and fast way to create monitoring for static systems, but now the trend is to create dynamic and scaling systems. This means that monitoring with static files is no longer the best way, at the static files needs configurations almost daily. Therefore a new solution is needed where the monitoring can be changed instantly without having to alter the static part of monitoring. This is where the monitoring-as-code principle comes to play. With this type of monitoring the idea is to add code to the monitoring system rather than alter a static file. This means that if the system that is being monitored scales out of proportions adding a new script to the monitoring will ensure its continuous operation. For this to work the monitoring tool needs to be built on code elements rather that static configuration files. A monitoring tool that completes this operation today is Sensu.

2.2 Sensu

Sensu[2] is an open source monitoring framework created by Sean Porter and sonian.com. It is often described as a monitoring router. Sensu operates with multiple "check" scripts and "handlers". The check scripts runs on multiple nodes and collects data, for example if Apache is up or down. It can also be used to to collect metrics from software like MySQL, after the data is collected it is routed to the handlers. The handlers job is to

determine what to do with the data generated by the checks. Both checks and handlers can be programed in any language which gives freedom for the users.

Sensu is made up by multiple small components that work together in the task of monitoring. Each component has a key role and task to complete to make it a complete system. The first part is the Sensu-server, which has the task of initiating the checks mentioned earlier. After initiation, the server also receives the reply from the checks and feeds them to the handlers. The server can also, from version 0.9.2, handle checks executed by clients without the server knowing. A single client can run multiple checks designed for specific purposes. For the communication between the server and client, Sensu uses rabbitmq for passing the data. It utilises SSL and the data gets encrypted before sending. The next component is Sensu-client, which runs on every node that is being monitored. Clients will execute checks scripts and return the results to the result to the server via rabbitmq as mentioned earlier. The last components are Sensu-API and the dashboard. The API provides access to data in the Sensu-server in Redis, and is usually run on the same server as the Sensu-server. Sensu-dashboard is a web GUI view of the current stat of Sensu infrastructure and gives the possibility to perform actions in the system and gives a good overview. A layout of Sensu can be found in figure 2.1.

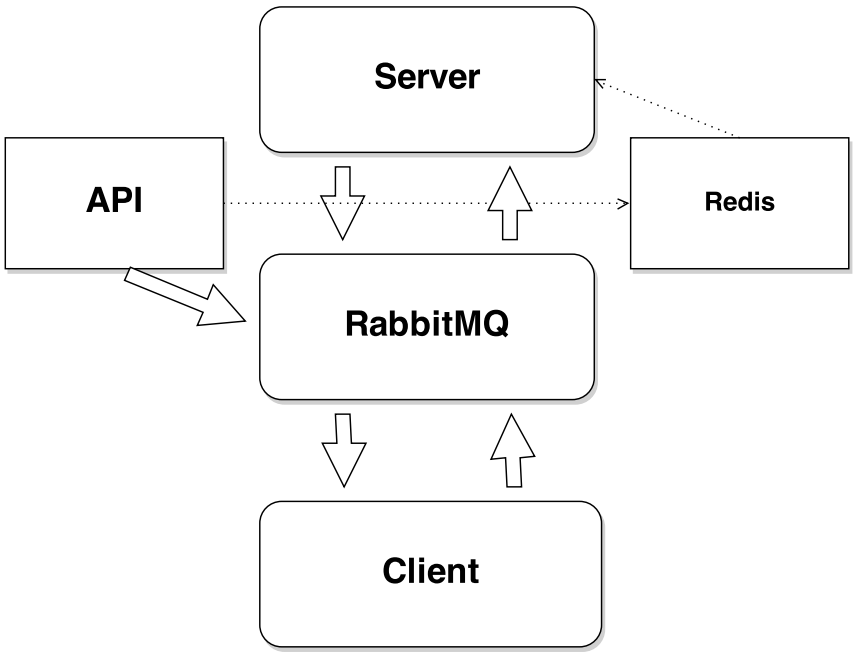

Figure 2.1: Overview of Sensu monitoring system

To configure Sensu it is important to be familiar with all the components mentioned earlier, so you get a better understanding of how this monitoring system works. Important to also mention is that Sensu can be used a monitoring-as-code where you can write you own scripts and plug-ins to customise how the program works in addition to exciting functions. This is an important part as it means you stand freely to choose what information to detect from the monitoring or create as an addition.

2.3 Information vs noise

Monitoring means as mentioned earlier to be aware of the state of the system. This means to collect all the information sent to the monitoring and evaluate the systems true state. When collecting such information there will aways be noise that interferes with the results. This can corrupt the results from the monitoring and lead to many false positives that's never wanted. On the other hand if you try to silence all the noise that might occur, it can also silence important alerts from the monitoring. Therefor there is important to find a balance between information and noise to maximise monitoring performance. To make this sorting better this experiment will look at the possibilities of using mathematical algorithms to distinguish between information vs noise. An example of a mathematical algorithm is EventRank.

2.4 EventRank Algorithm

To be able to solve this thesis there is a need to use an mathematical algorithm. The EventRank[3] algorithm is designed to rank individuals in a social network based on their participation in collaborative events. It was presented by O'Madahain and Smyth and was not originally designed for monitoring purposes, but rather rank individuals in a social network. The way EventRank works is giving each element a potential based on their participation in a network relative to the level of participation of other individuals in the same network. As participants contribute they get a higher scoring, but also loses point for each time they do not participates. The total amount of potential is preserved and can not be more that 1 or less than 0. EventRank is therefor very suiteable for real-time analysis of events that occur rather than using an algorithm for end results analysis. It is also argued by its authors to give a simpler output from the results than for instance PCA algorithm. The EventRank algorithm is displayed bellow with elaborations[4].

$$e \in C_i : R_{i-1}(e) + \alpha_i \frac{\bar{R}_{i-1}(e)}{\sum_{d \in C_i} \bar{R}_{i-1}(d)} \tag{2.1}$$

The first part of the algorithm calculates the new potential for events that participates in the round. Here the potential of event $e \in E$ at time t_i by $R_i e$ which takes on values from between 0 and 1. All events start out with the same potential $\frac{1}{|E|}$ where $|E|$ is the number of events in E. The combined potential will always remain the same and once an event reaches 0 it can not give away any more potential to others.

$$e \notin C_i : R_{i-1}(e)(1 - \frac{\alpha_i}{T_{N_{i-1}}}) \tag{2.2}$$

This part of the algorithm calculates the potential for those events that did not participates on the round. Each impact from the round is adjusted by α_i where $0 \leq \alpha \leq T_{N_{i-1}}$ and $T_{N_{i-1}}$ is the total amount of potential held by the events not part of the current round C_i. $\bar{R_{i-1}}(e)$ denotes the revers of the potential of e, i.e. $1 - R_{i-1}(e)$.

$$\alpha_i = f T_{N_{i-1}} \tag{2.3}$$

The last part of the algorithm calculates α after EventRank's own literature and defines it as figure 2.3. Here f is a constant that influences the dynamics of how fast events rises and falls. In this thesis the constant will be set to a slightly below-medium of 0.4 which makes it possible to do adjustments. In this thesis using the algorithm lead to a transient and cumulative potential for each event. The transient potential will be used to look at the real time status of each event. This score can be used together with the monitoring system to create alert and determine how important the event really is. The cumulative potential will show the summary of the score for the whole duration and can be used as a more historical look back of the testing period. The combination of the two score will hopefully help improve situational awareness in a monitoring system.

Chapter 3

Approach

This chapter will look at how the project will be conducted. First there will be a planing part where prototypes will be created and evaluated. Next there will be a part about what experiments will be performed and what they want to achieve. This is an important part as the approach decides what's going to be done in the rest of the thesis.

3.1 Prototypes

For this experiment to work, Sensu and EventRank need to work together and produce data that will give valuable information for the user. There are three ways of doing this operation, 1 either in real time where the data is processed at once, 2 with a delay where a cluster is processed or 3 post-processing where all the data is gathered and processed later. All these options need to be explored in order to see which is the best for this thesis experiments. To be able to gather data for this experiment, there is a need to produce false error messages to Sensu. This will be done on the clients, which again sends the messages to the server where it is handled. The false error messages need to switch on and off to make the EventRank algorithm work correctly. After Sensu receives the data from the client it has to be managed by EventRank. This can be done in different ways, as mentioned earlier, and this is where design of prototypes comes to play. Before choosing a way to conduct the experiment, all the possibilities need to be explored.

The first prototype that is going to be discussed is *Real Rime Processing* of the data also called prtotype 1. In this prototype every event that is gathered by Sensu is given to EventRank and processed. This meaning that EventRank has to be given every single event that occurs in the system. This results in a process heavy method because EventRank has to process every single event, and this requires a lot of machine power. The output from this prototype would be slightly delayed by the processing compared to running the setup without EventRank. When using this prototype you would only receive one output from the monitoring, containing the ranked results. This type of setup might be wanted for system that requires fast

feedback from the monitoring, and constant surveillance. The layout of this prototype can be seen in figure 3.1.

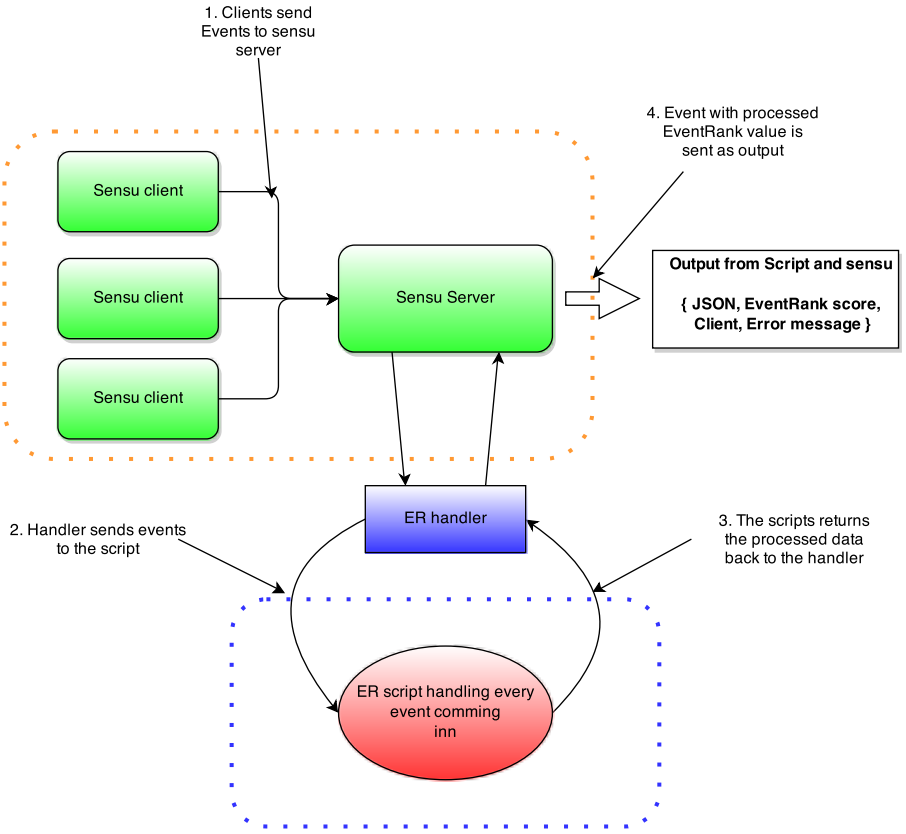

Figure 3.1: Overview of prototype 1

The goal of this prototype is to get real time output from the setup for those systems that requires constant monitoring without any delay. By constant output from this prototype, it is possible to get a better understanding of the events happening in the system and to act more accordingly to this information instantly for system administrators. As mentioned earlier, there will be a small delay for the script to process the data, but this will not affect the performance notably. Though this kind of setup is suited for monitoring, it is not the best way to use EventRank to calculate the score of each event. This is because EventRank needs to know all the participants in the system in order to give the most accurate score to each event, which is not possible in this prototype.

The second prototype is the one that is going to be called *Delay* refered to as prototype 2. The reason for this is, that in this prototype the data received from the clients are stored in a database and only processed in intervals by the EventRank algorithm. This gives a delay on the output received from Sensu decided by how long the interval is. In this way

it is possible to cluster events together and get a better understanding of how different events occur at the same time. The intervals can also be set to overlap one another, so that there will be no gaps in the information provided by the clusters. For this model to work, there is a need for a database, several scripts and a handler programed in Sensu. Though there are more components in this prototype compared to *Real Time Processing*, there is more room for adjustments on intervals and cluster size. *Delay* combines the speed of prototype 1 and the complete understanding from prototype 3. A layout of the prototype can be seen in figure 3.2.

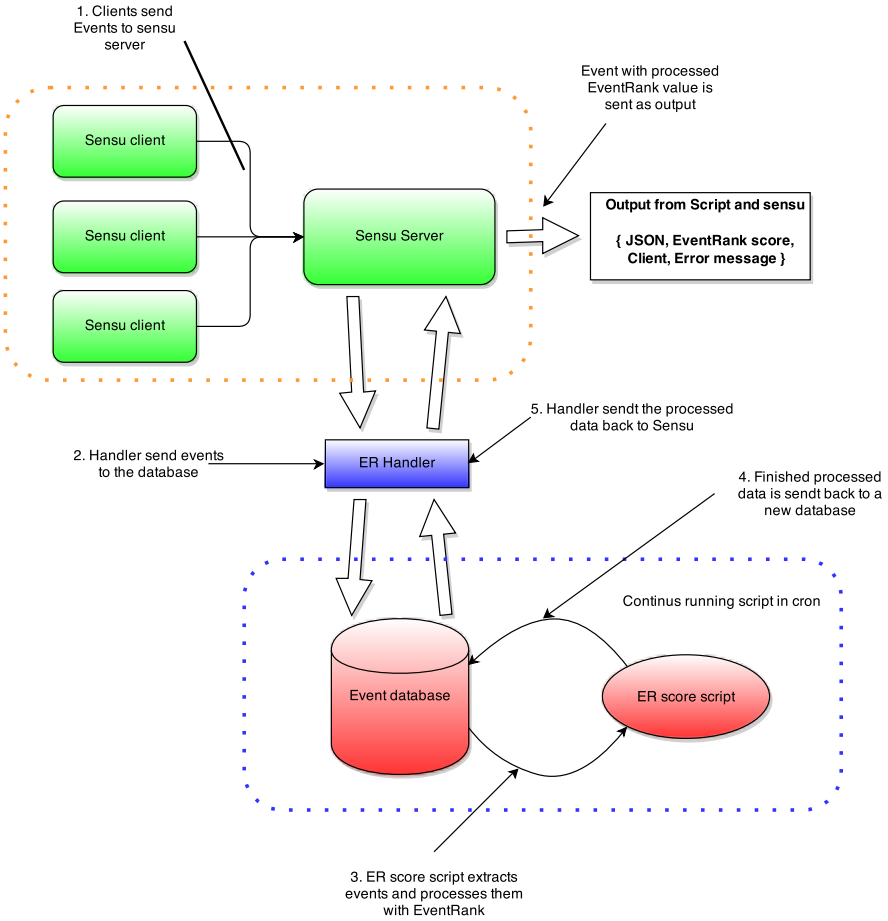

Figure 3.2: Overview of prototype 2

As the figure 3.2 shows, the setup of this prototype is a bit different from the first prototype in it's complexity. As mentioned, there is a need for an extra database and another script to make prototype 2 work. The goal of this prototype is to get a fast feedback from the system, but also to use the full potential of the EventRank algorithm. This is because it is now possible to give EventRank more events to calculate the score from the database, but still not holding back the results too long from the monitoring system

output. The process intervals is decided by the script and therefor it can be varied as seen fit by the user to suite the system. By solving the task in this way, it provides great flexibility, and it can be scaled as the system increases and different problems occurs.

The third prototype is called *Post-Processing*. Here all the data is gathered in a database and later given to the EventRank algorithm. The idea here is not to use EventRank to build a better real time output from the monitoring, but rather analyse the data later to see the coherence that might occur. In this way you will receive two separate outputs from the monitoring where one is regular monitoring output, and the other is from the EventRank. This type of solution of how to use EventRank can be helpful for organisations who want to evaluate a more overall picture of their system, without having to commit actions right away. For prototype 3 to work there is a need for a large database to store all the events from the monitoring and a script that handles the operation for EventRank.

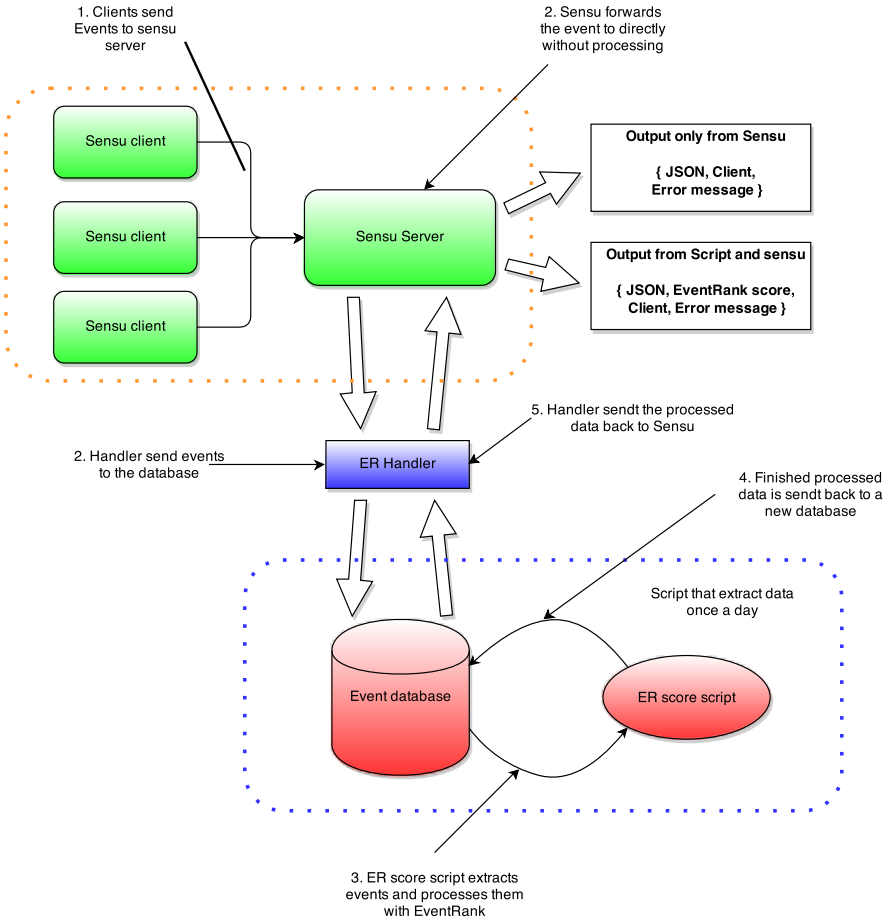

Figure 3.3: Overview of prototype 3

Figure 3.3 shows the layout of prototype 3 and its resemblance to the second prototype. Although it looks very similar it operates in a different way by storing all the events from EventRank in the database and processing them once a day or even further back in time. This means that the output can only be used to get a more overall picture of the situation in the system, and not as an addition in the monitoring system alert messages. Therefor, as shown in figure 3.3, it is two outputs from the processing, one with events from the monitoring system and another with the score from EventRank. This is why Sensu still can be used as a monitoring system, and why the additional information from EventRank provides a better picture.

Now that all the prototypes have been mentioned, it looks like the prototype 2(Delay) is best suited to solve the problem statement of using EventRank to improve situational awareness using a monitoring system. This is because it combines the best from both prototype 1 and 3. It is possible to adjust the intervals to fit EventRanks functionality so the system runs optimally and still can be used as a monitoring system that gives feedback to the users.

3.2 Experiments

To prove that EventRank can be used to improve situational awareness, this hypothesis needs to be tested. These experiments will determine if using mathematical algorithm helps understand what is going on in the system. To see the effects, there must be controlled parameters in the system and give the possibility to control whats going to happen during the experiment. Therefor there will be created a script that has the possibility of turning on/off 100 different error messages to Sensu. This provides a controlled environment needed to achieve results that will prove if the thesis can be used to implement in every monitoring system out on the market.

The first experiment that will be performed is to see if a single event can be distinguished from the rest and achieve a higher overall score by the monitoring system. This experiment will investigate the results of having this single event receiving all the extra score, and letting the remainder of the events lose score over time. This will be performed by turning on and off a single event for every interaction of testing, so that it will show up every time. The expectation of this experiment is that the active event will receive a higher rank than the others, and the others will also receive a lower rank in every iteration. This will be shown over time in the cumulative rank summary. This will prove that EventRank can find the event that causes the most trouble in the system.

The second experiment is quite similar to the first experiment, although a little more advanced. Here one event will start getting a lot of potential

first, but later a second event will take over and get a higher score. This will prove that EventRank can handle more events at the time and find the biggest trouble maker in the system. This will be shown in the graphs created from the experiment over time. Here there will also be used a script that creates error in the monitoring system, and therefor the outcome is controlled by the user. First, the script will create one type of error, and then create another. The expectation for this experiment is that one event will get alot of potential first, that flatten out on the graph. The other event will then take over and get a higher score overall. This will prove that EventRank can handle more events at the time and find the biggest "sinner" in the system.

These two experiments should be enough to prove that EventRank can be a useful supplement to a monitoring system, and help to get a better understanding of how the system operates. If there is time there is a possibility to do more experiments to better prove how EventRank works in more real life scenarios there will be performed a third experiment. This experiment would randomise every event that occurs and therefor it is not possible to control what is going to happen. The goal of this experiment is to determine how the system reacts under pressure and if it can follow the random inputs. To measure if the system performs as it should the result have to be compared with the random events that happened, the result therefore need to be stored for later comparison. This will prove the durability and performance of the EventRank algorithm and if it can be used in a more realistic scenario where one don't know what is going to happen.

This concludes the experiments that will be performed in this thesis. By conducting these experiment the results should be sufficient to determine if EventRank can be used to improve situational awareness in a monitoring system. The result will later be analysed to see if the experiments where successful and if the system can be implemented in a real life monitoring system. This would be the final test, but will not be performed here in this thesis.

Chapter 4

Results

The results chapter will look at how the thesis was conducted with regards to implementation and result. The implementation part will provide information about how it is possible to combine Sensu and EventRank into a single system. Some essential coding will be provided in the text and some can be found in the appendix. The result part will explain how the experiments where conducted and what results they have provided.

4.1 Implementation

Now that the prototypes have been designed, it is time to start implementing it into the environment. The environment that will be used is the openstack[5] cloud at HIOA where it is possible to create multiple virtual machines on the go. In this experiment there is only need for one machine, but it is possible to create multiple clients for further experimentation and even implement the experiment on control nodes in the cloud. By implementing EventRank on the control nodes, it can be used for monitoring on every VM in the cloud at school, but to start with it, will only be implemented on a single machine as a prof of concept. The machine is medium sized running Ubuntu 12.04 with Sensu installed, working as both client and server. The installation for Sensu is just a basic one without many checks and handlers, only those going to be used for experimentation with all important components installed.

The first thing that has to be created is the database needed as described in prototype 2. This is so it is possible to test the script containing the EventRank under creation. There will be needed three tables in the database, one containing the data from the checks and another that contains the processed data from EventRank. The third table will contain the accumulated score from EventRank to see the progression over time. All tables will have the events labelled with ID numbers and a timestamps from when they were created so it is possible to distinguish each event from each other. To create the tables, MySQL[6] will be used as the database tool

as it provides the functionality needed for this operations. A layout of the tables can be seen in figure 4.1.

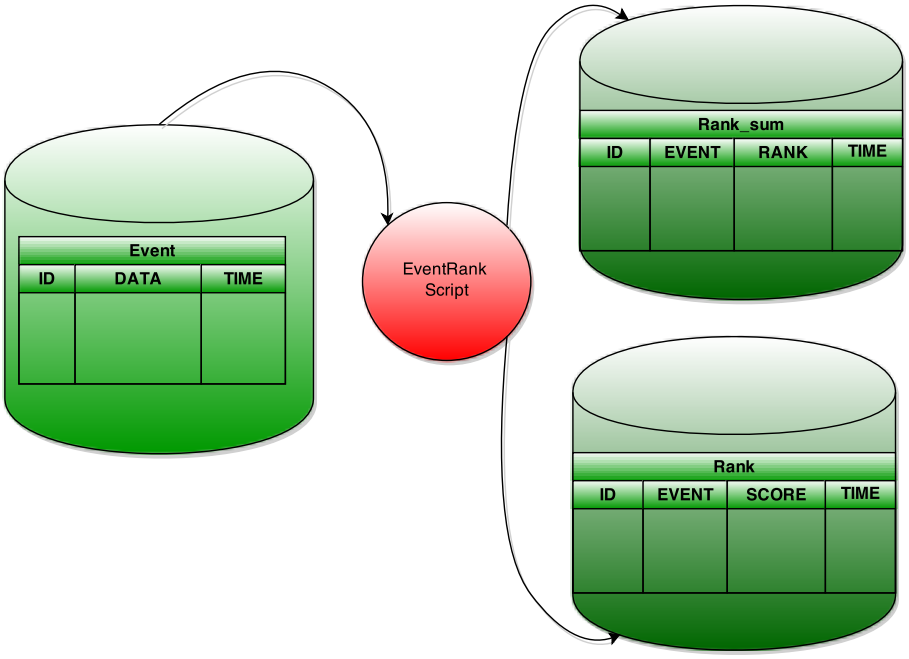

Figure 4.1: Layout of databases

As shown by the figure 4.1 the first table contains three columns and the second and third consists of four columns. They look similar, but the second one also contains the data from the EventRank script and the third the cumulative score. The reason for the three tables is so the data can be continuously processes without having to pause for writing operations to the tables and the data don't get overwritten. The first table will receive its data from Sensus handlers that continuously writes new events that occurs in the system into the table. After these events are processed by the script they get written to the second and third table before being extracted by the mutator created in Sensu. The databases will continue to fill up and therefor need some maintenance to empty old entries, but this can be solved will a deletion script or customised to only write new entries. In this experiment the quantities of data is smaller so it can be handled manually.

The next step of the implementation is to configure Sensu to send the events that occurs in the system to the database. This will be done with checks, handler and a mutator. They are all elements in Sensu that are designed for this purpose, but they need to be created from scratch. They can be written in any language, as mentioned before in the introduction and will be written in Perl. The goal of the handler is to fetch the error message from the checks and make the message readable to the output screen. The

checks are small scripts that only has a single function to report back to the system if the script fails. A strong feature in Sensu is that multiple checks can report to a single handler script for further processing of the data. This means that it is not as much work creating multiple checks and handlers because everything can be duplicated. The checks that will be used for this experiment is a simple test if a file is missing from the system. By creating 100 unique files in /tmp/ it is possible to create checks for each to receive up to 100 different events. This makes it possible to test out the EventRank algorithm in the environment where there is a fixed pool of events, and it is possible to turn the event on and off by simply removing or adding the files. The next step is to configure the handler and make Sensu receive all the messages from the checks. To complete this operation in Sensu there is a need for two files for the handler. One file called /conf.d/eventhandler.pl that tells the checks where to send their data. This file tells Sensu to send the data received from the checks to /handlers/erhandler.pl. This file needs a little more work to complete. Here the data is received and decoded with the JSON function in Perl before the output is printed to the log file. It is this data that is displayed to the dashboard and alerts the users of a mistake in the system. This will be used in the experiment as a way to tell if the errors have been detected by Sensu.

Now the data is picked up by the monitoring system, it is possible to start manipulations of the with the mutator. It is the mutators job to insert the data decoded by the handler into the tables in the databases and check if the event has occurred before. This is a more complicated operations that requires a bit more work scripting together. The first part is to check if there is a new event that Sensu is reporting on. This is done by printing all new events that comes in to a log file and then check if the new events matches with those already in the file. If the event does not match with the entries in the file, it is printed to log file and written to the table event in the database. In this way there will only be new events printed to the database. How the script is written can be seen in the code below. Now all the data from Sensu is gathered and decoded, and it is possible to start processing the events with the EventRank algorithm.

```
my $event = <STDIN>;
chomp $event;
$event =~ s/\R//g;
open(LOG,">>/var/log/sensu/eventrank.log");
my $hashref = decode_json $event;
print LOG time . " got line: $event\n";

my $occurences = $hashref->{"occurrences"};
my $status = ${$hashref->{"check"}}{"status"};

print LOG "Status: $status, Occ: $occurences\n";

if ( $occurences == 1 and  $status > 0 ){
    print LOG "Valid event, inserting into DB\n";
    my $dbh = DBI->connect("DBI:mysql:database=......;
    host=127.0.0.1","user","")
    or ( print LOG "Error: $!\n" and warn("error: $!\n") );

    if ( $dbh ){
        $dbh->do("INSERT into event(DATA) values('$event')");
        $dbh->do("SELECT SCORE from rank");
        $dbh->disconnect();
    } else {
        print LOG "no connection to db\n";
    }
}
```

This script will be the most important part of this thesis as it is where the score for each event is calculated and processed. The previous steps have made it possible to extract data from one database, process it and place them back in another database. The first step of this script is to extract the events recorded in the event database. Here it is important to make sure to only extract the last events that have occurred as this is a continuously running script. There is therefor a check in the script that monitors the last processed events ID number in a file, and only extracts event that is the successor to this number. An example of the check and extraction from the database can be seen in the code bellow, this show how the data is fetched and put into variables.

```
open(EV,"$ID_FILE");
my $LAST_ID = <EV>;
chomp $LAST_ID;
close(EV);

my %events;
my $event = $dbh->prepare("SELECT ID, DATA, TIME FROM
            event where ID > $LAST_ID");
$event->execute() or die $DBI::errstr;
print "Number of rows found :".$event->rows."\n";
while (my @row = $event->fetchrow_array()) {
    my ($ID, $DATA, $TIME ) = @row;
    $DATA =~ s/\R//g;
    $LAST_ID = $ID;
    print LOG "ID = $ID, Data = $DATA, Time = $TIME \n";

    print "ID = $ID, Data = $DATA, Time = $TIME \n";

    my $hashref = decode_json $DATA;
    $events{${$hashref->{"check"}}{"name"}} = 1

}
```

As shown in the code the check also adds a new ID number to look for next time the script is running. It has also printed to the log all the new events that have been extracted so it is possible to take a look back if needed. The next part of the script is to start processing the events with EventRank. As the script runs continuously, the first step is to extract those events that already have a score from the rank table. This means another MYSQL query from the rank table where the events and the score is extracted. This gives the possibility to recalculate the score of those events that already have been processed and give a score to new events. The variable extracted from this query is EVENT and SCORE and they are linked together where SCORE is dependent on the EVENT. Now that the new events from the event table and the old events from the rank table is extracted it is possible to start calculation the new potential for all the events. This is a two part operation as EventRank give more potential to those who participates and less potential to those who didn't participated in this round of calculations.

```
if ( %events ){
# calculate new potential

    # calculate sum of potential for all active
    my $sum_d;
    foreach my $e (keys %events){
        print "event was active: $e\n";
        # $rank{$e} += 0.01;
        $sum_d += ( 1 - $rank{$e});
    }
    print "sum_d = $sum_d\n";

    my $T_N;
    foreach my $e (keys %rank ){
        if ( not $events{$e} ){
          #   $rank{$e} -= 0.01;
            $T_N += $rank{$e};
        }
    }
    print "T_N = $T_N\n";

    my $f = "0.4";
    my $alpha = $f * $T_N;
    print "alpha = $alpha\n";
    foreach my $e (keys %events){
        print "event was active: $e ($rank{$e})\n";
        # $rank{$e} += 0.01;

        $rank{$e} = $rank{$e} + $alpha *
        ( (1 - $rank{$e}) / $sum_d);
        print "new rank $rank{$e}\n";
    }

    foreach my $e (keys %rank ){
        if ( not $events{$e} ){
          #   $rank{$e} -= 0.01;
            print "event was inactive: $e ($rank{$e})\n";
            $rank{$e} = $rank{$e} * ( 1 - ( $alpha / $T_N ));
            print "new rank $rank{$e}\n";
        }
    }

# update event potential in rank table

    foreach my $e (keys %rank ){
        $dbh->do("update rank set SCORE =
        '$rank{$e}' where EVENT = '$e'");
```

```
        $rank_sum{$e} += $rank{$e};
        $dbh->do("update rank_sum set RANK =
        '$rank_sum{$e}' where EVENT = '$e'");
    }


} else {
    print "No events this round, nothing to report\n";
}
```

From the code it is possible to see that the script calculates the score after the formula described in the section about EventRank. Here it states the perimeters of the formula and they have all been completed in this script. Both by adding and removing score from the participants as they partisipates. The first part is to calculate the sum of potential for all active members in the round. The next part of the code is to calculate the T value of the formula. This must be done in the beginning as it is to be used later on in the calculations for removing score for those who did not participate. After these two values have been determined, it is possible to start the calculations on how much rank the active participants will receive. This is done by using a constance of $f of 0.4, this value can be adjusted if the interval of events changes. The values are calculated for each event and given to the variables. The last part is to write the updated score to the table rank, so they can be used for the next round of calculations. This completes the calculations of the variables and the read and writing operations. The only thing remaining is to extract the data from the script to be able to perform analyses later. This is done with openTSDB in a small foreach loop that looks like this:

```
foreach my $e (keys %rank ){
    system("echo 'put eventrank.transient_potential $timestamp
    $rank{$e} event=$e' | nc 128.11.111.11 4141");
    system("echo 'put eventrank.cumulative_potential $timestamp
    $rank_sum{$e} event=$e' | nc 128.11.111.11 4141");
}
```

OpenTSDB(time series database server)[7] is an open source software that has the ability to receive large quantities of data and process them into a GUI that help get a better perspective. It uses databases to optimise time series data and can handle many thousand request every second. It is therefore a very useful tool for analysing the data output from the script. It also have the function to export the data to other software's to create graphs. Now all parts of the system is implemented and it is time to start the experiments.

4.2 Experiments

Now after the implementation is done, it is time to start experimenting. As mentioned in the approach there will be two main experiments where it is possible to establish the base for the success of EventRank. If there is time, it will also be possible to test the script in large scale in the cloud environment of HIOA. The first experiment is to find out if it is possible to find the event that occurs the most often in the system. This meaning that there will be a single event that occurs more often and should be given a higher rank than the others. This is against how EventRank was intentionaly meant to operate as it is to reward those who contribute and not those who don't participate. However it is possible to manipulate the results by reversing the order of how EventRank sees the environment there by receive correct results. The second experiment is to see if EventRank has the potential to distinguish between a primary event and a new event that will take it's place. This is to test if the script and EventRank can handle a more real life scenario. To test this there has been created simple scripts that run in the background to remove and adds files every 60 seconds. How Sensu finds these files has been mentioned in the implementation of the checks. The script can be customised to a number of different scenarios depending on the result wanted by the user, which is suited for the experiments that will be performed as they are a bit different in origin. Also before each testing of the system there is a need to update the databases by resetting the score to 0.1 and emptying the cumulative score table. This is so that before each test every event has the same potential and the chance to score equally good in the potential outcome. This together with all 100 file makes it possible to start testing the different experiments.

Experiment 1 as mentioned in the section before is to find out what EventRank will do when there is one repeatedly strong event in the system that should accumulate most of the score. As mentioned before all the scores and files are reset back to basic settings before the experiment starts. The first time that the system runs this kind of test is to see if it is actually possible to run a monitoring system with the additional functions like an mathematical algorithm. Therefor the first time experiment 1 was tested it was only a short test to see if the system catches the events. The result of the test can be found in figure 4.2 where a single event has accumulated most of the score.

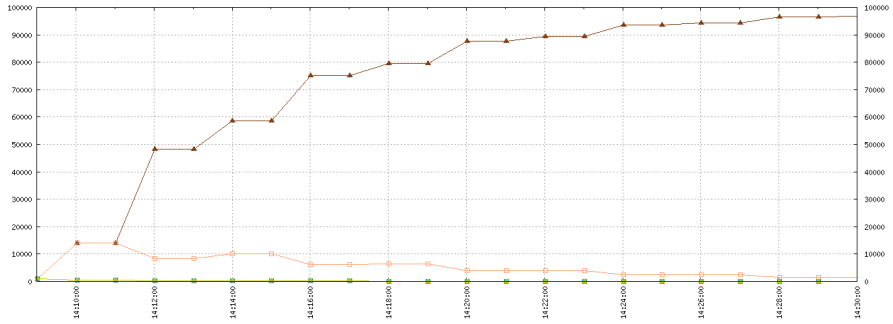

Figure 4.2: Experiment 1.1

As this was only a short test to see if the functionality of the system worked as it should, it was possible to extend this test. This meaning that experiment 1.2 could be created with more functionality and run for a longer period. In this experiment the script that removes and adds files. It will first mostly remove file001(event 1), then bringing the score back to the others before once again focus on event 1. The script will approximately run for 45 minutes. When a event is active for this lone it should have recived almost all of the score and be above 0.9 of the 1 possible. This will show if the system can give a single event can have most of the score, then distribute it back to the other participants that did not accumulate score in the first round of the experiment. This will be a more realistic every day scenario of real time events that has a high frequency. The transient results can be found in the figure 4.3:

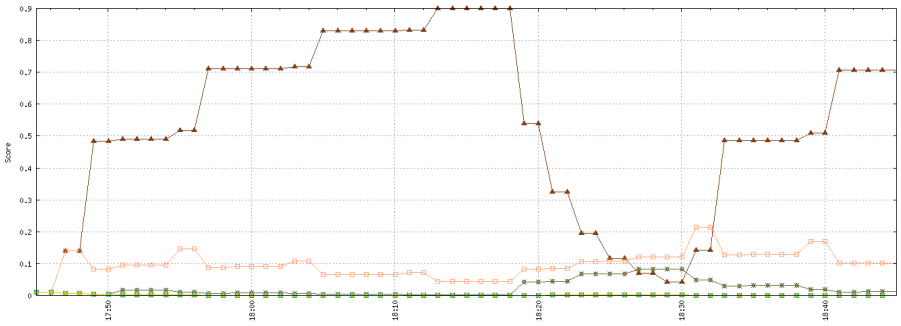

Figure 4.3: Experiment 1.2 transient graph

The experiment was a success as the system is able to give the correct score to the event and also remove the score again as expected. The figure 4.4 also shows event001 gets most of the score, which is correct according to the script that adds and removes file from the system as it should. At 15:59, all other events participates except event001 and it loses score to the others. This is where the graph drops sharply. Before the drop, event 001

reaches a score of 0.84 which is 84% all the score possible, which leaves only 0.16 to be distributed to the other 99 events in the test. At 16:11 the script is set to start focusing on event001 once again and therefore it starts to climb up again. This time it does not reach as much potential, but this is because the experiment did not run long enough. Had the experiment run longer it would have reached a higher overall score in the end. This shows that it is possible to use an mathematical algorithm together with a monitoring system to provide a better understanding of the operations in the system. If this had happened in a normal monitoring system it would not be possible to see the extent of the error created by event001. The graph bellow shows the cumulative output from the experiment above. It shows the sum of all the score event001 has gathered over the experiment time. Also this shows how event001 flattens out a bit before continuing to climb in the overall picture.

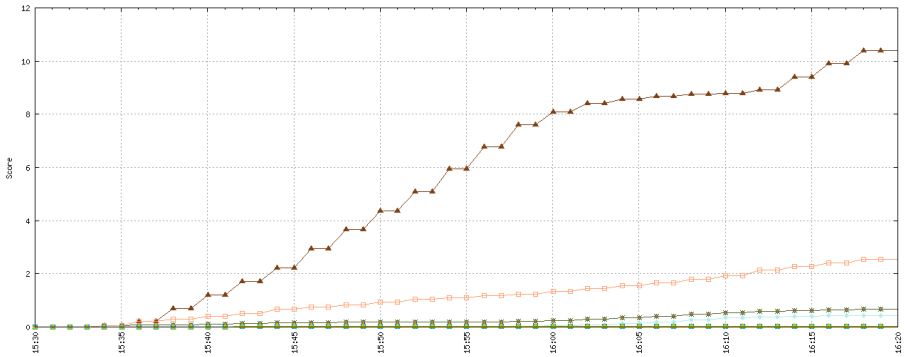

Figure 4.4: Experiment 1.2 cumulative graph

The next experiment that will be performed is experiment two. The goal of this experiment is to see if the system can handle distributing loads on different events. Here one event will start and another will take it's place as the experiment goes on. Here the script that will add and remove files from the system will start off by focusing on event001 and transition to even004. Also event002 and 003 will be a part of the test as a control group to see the overall score of semi-participants. This experiment will run for approximately two hours before it stops. Another thing to look at in this experiment is to see how much the control group is affected during the transition between the two events. The result from the experiment can be found in the figure 4.5

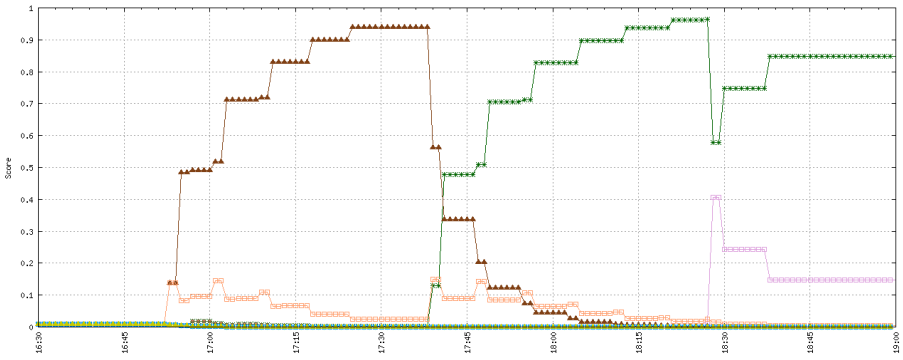

Figure 4.5: Experiment 2.1 transitive graph

As seen from the results from the experiment it was a success. Event001 started in much the same way as experiment 1, but was overtaken by event004. The graph shows that this happened in only a 4 minute window when the experiment was run in this way where there is only focus on a single main event. In this experiment it also shows that the curves flattens out as time goes, as the other events have less and less score to give away. The transition happened at 17:38 where event001 has a score of 0.94 before dropping bellow 0.34 within 4 minutes. This is because of how EventRank is designed with sharing the score of those who have it. When an event has a high rating, it stands more to lose if it don't participates in a iteration. As event001 losses score you can see that event004 gains the same amount in return, as non of the score can be lost and has to go to the event that is active. This is mirrored throughout the test as it progresses and event004 reaches an score of 0.96 before the test is over. By this time event001 has declined so much it is indistinguishable from the other 97 unimportant events. The experiment also shows a small peek from event007 that had some potential during the testing. The figure bellow shows the cumulative result from the experiment. The control group also gets score over time and performed as intended. The graph shows that it climbs steadily throughout the experiment.

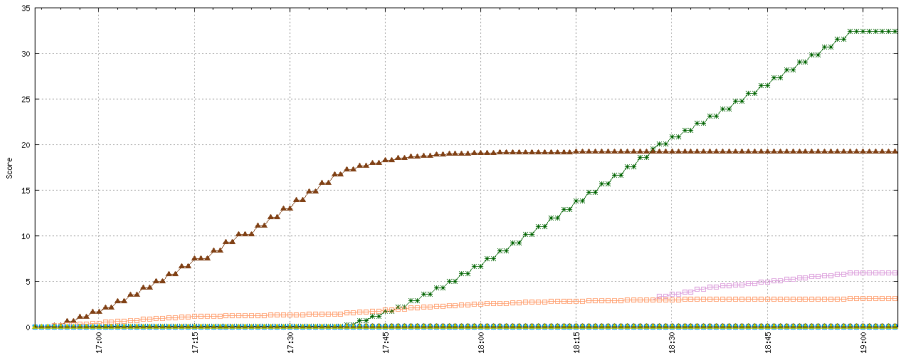

Figure 4.6: Experiment 2.1 cumulative graph

The cumulative results from the experiment shows how event001 gains a lot of score in the beginning of the experiment and the flattens out in the curve as it is not gaining that much score anymore. This is because of how the algorithm works by only giving a low score to those who don't participates over time, thereby not letting them grow that much on this graph. Event004 then takes over and climbs beyond event001 as times go by as it is the most active during this time period. It keeps climbing to the end of the experiment as it should according to the script that adds and removes files from the system. An interesting finding from this experiment is that there's almost impossible to see the other event when one event gets a really higher score over time. This is something that needs to be looked at in further experiments as it might turn into a problem over time.

Now that the main experiments are done and there is still time left to do another experiment where the parameters are not fully controlled to really put the system to the test. By this it means randomising the events that alerts the system. This is a good way to really test if the system can handle a more realistic load from a real life situation where you don't know what's going to happen. To make this happen there is a need to crate a script that pics a random numbers between 1 and 100 and removes the file with the same number. To be able to replicate the results and do further analyses, the numbers picked in the experiment and how often they where used need to be stored. To solve this problem there will be created a script that has the ability to create another script that can remove and add files. In this way it is possible to recreate the results by running the same script twice, and also possible to make statistics on the numbers used. The first script also can control how many interactions there will be in the experiment and thereby how long the script will run for. Also to make it really random the number of events in each iteration are also randomised and can be controlled also in the first script. The first time the experiment run it used 50 iterations with up to 50 random numbers that could be picked. The number of occurrences of each event can be found in figure 4.7.

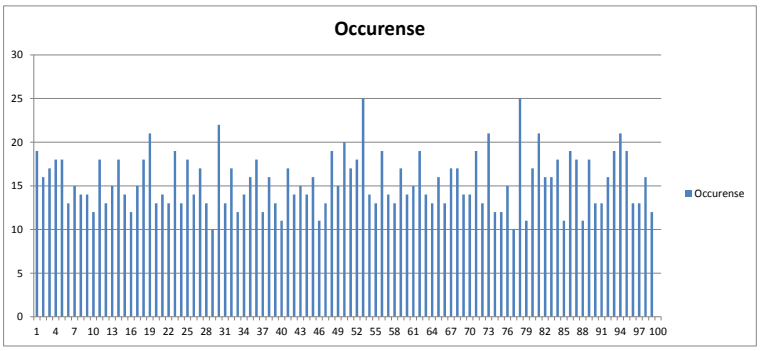

Figure 4.7: Number of occurneces

As shown from the graph the numbers of participation is evenly distributed between the events with some events getting a few spikes above the others. It is particularly two events that stand out, 53 and 78 which bout have 25 occurrences in this round. This is approximately 10 more happenings that the average of 15,36 and a percentage of 0.6 % more than the others. This is something that EventRank should pick up in the experiment and might give these events a higher overall score in the cumulative graph. Although these events happen the most the score in not

calculated on number of participation's, but on how many participated in the round. Therefor it will be interesting to see if it is possible to distinguish these event from the rest through the experiment. When the experiment is running, it looked quite different from the other experiments as now there is not a single point of focus, but spread out on all the events. Right from the start the score begins to spread out between those who participates. As seen from before in the other experiments how much each event grows or falls depends on how many participates in the iteration as mentioned before. This is also why the scale of the graph is much lower than before as more events participates and there there is not a high peak as before. As time goes more and more events have gotten a score and the results become a bit disorganised by the looks of it seen in figure 1.14. Although when comparing the results in the graph to what events where triggered compares to the events that was supposed to be triggered in the script so the functionality works. This shows that EventRank works in this type of settings and can be a use full part of a monitoring system. When looking for event 053 and 078 they don't particularly stick out in the graphs as they mostly happen together with other events and therefor is hidden amongst the rest. However event 053 gets a high peak at 18:58 and event 078 at 19:04. Another interesting peak in the graph is between 18:52 and 18:53 where a single event(event086) scores most of potential by being in a small group over two interactions.

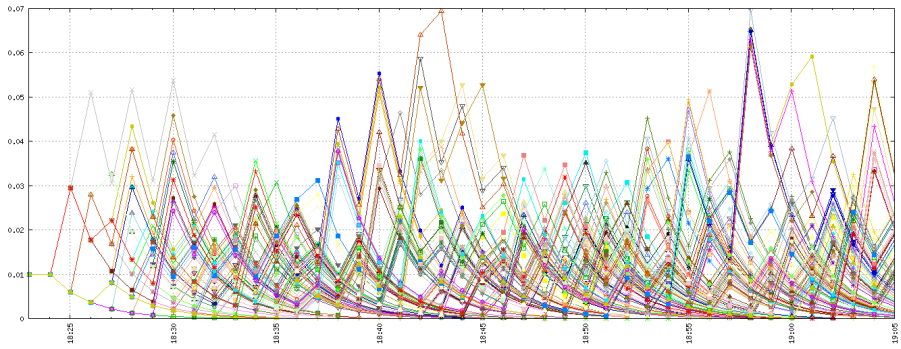

Figure 4.8: Experiment 3 transitive graph

When we look at the cumulative graph it shows how the events gather score over time. This should mirror the number of times they participated in the second script, but also has the addition of the calculation from EventRank. As seen from the graph there is a main group of events that have accumulated a score between 0.27 and 0.45. This main group is more that 75% of the events and is also where the average of the score is located with 0.399. An interesting result from this graph is that neither event053 or event078 has the highest collected score. As seen from table ..... they are both top 10, but have a lower overall score with the places 5. and 8. The reason for this is eventough they appear the most, they also share their

appearances with many others at the same time so they don't accumulate that much score. The highest score bellongs to event 030 which had 22 appearances in this experiment and had 0.1631 point higher score than the next. This shows that the results from the experiment is a bit different that expected at the beginning, although not surprising to find.

Place	Event	Score	Number of participation
1.	event030	0.9792	22
2.	event086	0.8069	19
3.	event019	0.7836	21
4.	event053	0.7402	25
5.	event014	0.7220	18
6.	event005	0.7151	18
7.	event059	0.7097	17
8.	event078	0.6640	25
9.	event076	0.6561	15
10.	event071	0.6402	19

Table 4.1: Table of 10 highest cumulativ scores

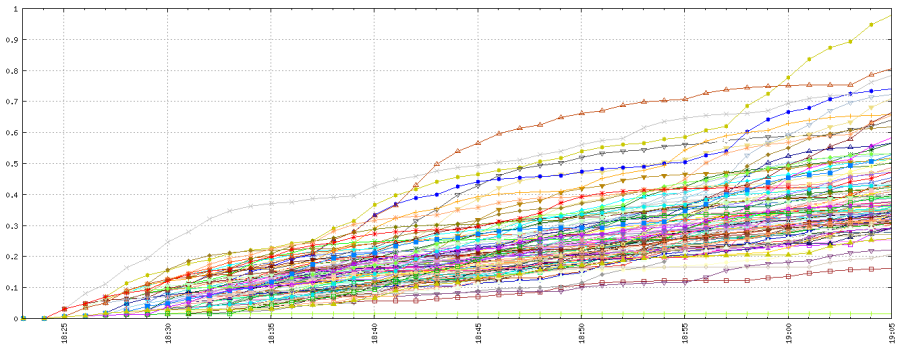

Figure 4.9: Experiment 3 cumulative graph

This concludes the results done in this thesis. Some of the result where expected, but others where a bit unexpected as shown with experiment 3, where all the events where picked randomly. All the experiments where successful in creating results and helps understand the impact EventRank can bring to a monitoring system.

Chapter 5

Analysis

5.1 Analysis

The first thing to look at is how the implementation worked in this project. The implementation was where most of the work in this project was done to make it possible to test if the problem statement was achievable. The implementation itself consist of in addition to Sensu, 3 tables and two custom scripts. This was the same way as mentioned in the prototypes and the design was carried out in the implementation. All of these extra functionality needs to run to be able to use EventRank as an addition to Sensu. The implementation was a bit tricky in the beginning, but when fully understanding the functionality already in the system the work speed up a great deal. When it comes to performances, the database can be performance insensitive. In this system there is only small read and write operations in this implementation that don't take a lot of performance from the system. These operations will take more and more performance from the system as the scale increases, but this will not normally be a problem as monitoring system is normally performed on a separate machine. Also the scripts that extract and compute the data are not very CPU intensive as they run in the background and the performance of the is not notably effected by this. These script also only run a short time in the intervals set for them which only give small spikes in the performance and don't affect normal operations. All in all the implementation was a success and it can be used in a large scale system which was an important part to achieve in this thesis.

When it comes to Sensu as a monitoring system and implementing the extra functionality, it was also a success. This is because Sensu has the support for adding self defined code to the functionality, as mentioned in the introduction monitoring-as-code. Since it utilises monitoring-as-code it is very suited to this task as it has basic functions, like checks and handlers, that can be customised to every need. This made it possible to extract the information needed to perform the calculations for EventRank. Not many other monitoring systems has the same functionality and it would have made it much harder to complete this thesis. Although Sensu is suited for

the tasks, it took quite a bit of work to make it possible as entire parts of the functionality had to be coded from scratch. This is quite difficult if you never have used this monitoring system before as you need multiple small scripts working together. A last part of the Sensu functionality was never completed with sending the processed results back to the dashboard for a more user friendly interface. This would have made the monitoring system complete and easier to use for people who don't have created the system.

Before the analysis of the result there are some factors that need to be mentioned. The first being that there are 3 factors that decide if this implementation will performe well. The first being the number of events happening at once. This meas that the more events happening at once, the heavier the load get for the system and it has to work harder. In the experiments this was never a problem, although it was not monitored during testing. The second factor is how many events are in the system. This decides how the rank is calculated and how much rank each of the individuals gets. EventRank has to max have a total score of 1, and therefor if there is 1000 different events, each would start with 0.001 in score. Both these two factors is something that can't be controlled by the operators because system always change over time. This gives a bit of a problem to the current system as these two factors have been controlled the entire time and it is been accounted for. The third factor is the interval in which the rank is calculated. This can be controlled by the operators and customised to the system it is going to be used on. This is an important factor to be able to control as it needs to be set according to the intervals your system operates on. As fast feedback system, short intervals. Slow system, long intervals.

5.1.1 Results

The results in this thesis was as mentioned both expected and unexpected. The first experiment the results where as expected with a single event that would dominate the whole group. This result proves that EventRank can be used to not only find the biggest error in the system, but also show how much disturbance it has caused over time. This can be very useful if one is wondering where to start error searching in a bigger system and need a starting point. This test also proved that the implementation worked and more extensive tests could be performed. As seen from the results event001 quickly climbs in the beginning as it accumulates almost all of the score available in the system and flattens out as the other events don't have more score to give. A problem that this test revealed is that it is almost impossible to distinguish between the other events as they have very little score and become almost hidden at the bottom of the graph. This might be a problem if you want to see how the other events are doing without extracting the information from the database. Although this is the result from this experiment, this will only be the result if a single event takes all the score over a longer period of time. This also inspired the introduction of the cumulative graph that collects the combined score over time.

The second experiment was a bit more complex where one event would start as experiment 1 and the be overtaken by another event after a period of time. This was to show what would happen if more that one event was active in the system and if the implementation could handle processing more scores at once. By conducting this test it also proved that events could come and go and the system could run for longer periods of time without have to be reset. The results show that the scores climb as expected and drop of when they don't participate in the active group and it was a successful experiment. This shows that the implementation can handle more advanced settings and be able to withstand heavier loads that can produce more realistic data and opens for more testing. Experiment 1 and 2 shows the functionality that was supposed to be proven in this thesis and displays that an mathematical algorithm can be used as an useful addition to a monitoring system. Although this does not reflect a normal monitoring environment where anything can happen, and additional tests where needed to be performed to prove this.

The third experiment was conducted as mentioned before to find out how the implementation functioned in a more real time scenario where the circumstances are not controlled and therefor end up with a random result. This will show how the implementation handles multiple events over a longer time period and how accurate the results are when they later compared to how many times each event appeared in the script. From the start to the end, every event was picked up by the EventRank script and processed to the database. It proves that the implementation can handle large loads and be able to take unexpected surprises that might occur as it is not controlled in any way. This also proved that the system is accurate and can be trusted to catch everything that goes on in the system whatever the situation. When it comes to the cumulative result compared the number of appearances in the system there was a surprise that the events that didn't participate the most accumulated the highest score. Event053 and event078 both had 25 appearances, but ended up on 4. and 8. place in the overall score. This shows that eventhough they appeared the most, it was always with other events, and therefor don't get a high score on their intervals. EventRank was designed to do exactly this by sharing the score over all the active individuals and it therefor adds up to the results received by the test. This shows that the system act as intended and it is possible to find the events that stick out the most from the rest of the group. A negative side is that you need this knowledge about how EventRank operates to be able to find the events that is the biggest sinner. This was the last test conducted, but to relay prove that this system works, it needs to be tested in a real life system with real events.

Chapter 6

Conclution

The assignment is now done with the experiments and the analysis of the result. It is time to start discussing what could have been done differently and if something could have been done better. This chapter will look at this and if this thesis was successful by assessing if EventRank can be used as an addition to a monitoring system. Also an important part of this is if EventRank and the system can be customised to fit any infrastructure to prove its usefulness.

6.1 Discussion

After this project was done it was clear that the system created was a bit complex in the way it was created. It consist of multiple scripts and tables in the database. This is something that is a bit difficult to comprehend for people that don't have created the system and should be simplified. For instance instead of having 3 tables it is possible to combine rank and rank_sum and therefore only have 2 tables in the database. This will lead to a bit less configuration and be easier to comprehend when trying to get an overview of the system. When it comes to the scripts they have been kept to a minimum and there is not much that could be removed without risking to lose some of the functionality. One thing that could be changes is that the scripts not only process the the newly arrived events, but also the events that have been processed to create a overlapping interval. At this stage there is no such functionality and the prototype suggested that it should have been in place. It is a good thought to have this functionality within the system, but it is not really needed as the current setup gives good results, and don't miss events that have happened. This is something that could be created at a later stage to see how this functionality would affect the system. The last part about the configuration the need to be mentioned, is that the processed data is never transmitted back to the Sensu dashboard. This would have been a very useful feature to the system that was not implemented as there was not enough time in the end. The reason for it not being prioritised was that it is not necessary to prove whether or not EventRank can improve situational awareness in a monitoring system.

The next thing that need to be discussed is whether or not EventRank can improve situational awareness in a monitoring system. This is the most important part of this thesis as it shows if this was a success or not. What has been proven is that the system has the ability to find and extract a single event from the others and also capture if there is more than one active event. This shows that the Sensu with the collaboration with EventRank can distinguish between the trouble makers and those who only have one or two errors. This is very useful when trying to find patterns as they will show up in the graphs and you can clearly see that they are part of the same problem. This is a strong argument for the problem statement as it gives a better oversight over the situation and it is easier to start trouble shooting for the problem. Compared to a normal monitoring system, the events would only be visable when they where active and not possible to find the same patterns. Another great thing about the setup, is that you can also go back in time to identify if the same has happened before with the same events. This will give a system administrator good evidence if something needs to be changes in a system as it can prove tendency, and patterns that can be hard to find otherwise. Also the cumulative scoring system is good to capture if an event is unstable and causes many alerts, that can be seen as false positives. They will then accumulate a high score that can be spotted in the graphs, and take preventable actions against in the future. Experiment 3, where all the events where totally random also shows that the system can handle heavy loads and sort the events as they happen without mistakes. All though this shows the durability the system has never been proven in a real life scenario and this is something that needs to be done before one can really confirm this as an useful edition to a monitoring system.

The last part of this discussion will be about EventRank it self. As shown from the experiment, it is possible to create a system that can use this algorithm and calculate the score of the participants. To be able to to this there where some parameters that had to be, for instance like the number of event and how much score each event would start out with. This is not something that can be done in an scaling system as these parameters would have to be changed every time the number of events differentiate. There is therefor a need to automate this operation where these two score will automatically be adjusted by itself. Without this it would be impossible for EventRank to calculate the correct score for each event. This is something that should be possible to create without to much work as it is simply a calculation over how many events are present in the system at any time, and never going over the limit of 1 in the combination of the scores. When this is created it should be possible to fit this system to any monitoring system and help system administrators make better decisions. The last thing that needs to be mentioned about how EventRank operates is that it only reacts to new event in the system as its natures intended. This is a problem if a single event is active for a long period of time, as it will not receive a higher score even tough it is still active. The event will then,

in worst case scenario recive a so low score that it is not visible amongst all the other events, and therefor not alert the system administrator. To solve this problem it is possible to create a script in Sensu that will remove the event and add it again after certain period of time. This will allow the event to accumulate score over time and not be hidden amongst the other events even though it is active and causing problems. If both these potential problems are solved this system will be an great addition to any monitoring system as it will help catch trends and tendencies in all system and save time for system administrators all over the world.

6.2 Further works

When it comes to further work that can be done in on this thesis there are some point that should be mentioned. The first being to implement this system with EventRank in a real system where there is no controlled circumstances and anything can happen. This is so that the system can be tested under normal condition to really prove its functionalists. This would be the final test to sees if the system gives as good feedback here as it has done under controlled testing. This would also be a good time to test the compatibility of the system as it would most likely be tested in a somewhat different environment where it has to be adapted and maybe discover flaws in the setup. Also to improve the user friendliness of the setup it should be adding the feedback from the system to the Sensu dashboard. This would be useful for the users and also reduce some of the complexity of the system by eliminating the need to use openTSDB to produce the graphs. A goal for this operation should be to gather most part of the system to a single setup without having to rely on other components.

What also should be considered to if this thesis is continued is adding more functionality too the setup. By doing this it would improve the usability and also make it more attractive to other users. On such functionality could be to arrange the processed events into different categories depending on how much score they accumulate. Thereby when a event reaches high priority this is something a system administrator would have to look instantly and thereby save time prioritising. As an start there should be three classes, low, medium and high. Another useful addition would be to add groups where events are monitored together. This would be useful to detect patterns that are abnormal and alert on this also. If one of the participants in the group malfunction and not the rest this would be useful to know and maybe take a closer look at thereby creating a better situational awareness.

6.3 Conclusion

As the introduction stated there is many problems in the daily operations of an system administrator and amongst them monitoring plays a big part. To configure monitoring correctly can be an demanding task and sorting out the information from the noise is an especially difficult part of it. To help doing this thesis was designed to create a better monitoring system that could improve situational awareness for system administrators by providing more useful information. By achieving a better monitoring system a system administrators can focus more of their work on development of the system rather than searching for the errors that have occurred. This searching can be performed much faster if the information of patterns and where the problem occurred is displayed in an orderly matter. This is where EventRank come into the picture, as it has the capability of processing events and giving them a rank calculated on their participation. As shown in the thesis this was achieved by implementing the formula into the monitoring system Sensu. These two parts combined managed to provide a better monitoring system that has the ability to give users more helpful information when needed. It has the abilities to accurately extract out those events that causes the most disturbance and also provide information about what else was going on at the same time in the system. This gives the users a better picture compared to only get an error message about every single event that occured sepratly.

The implementation from the project shows that it is possible to integrate Sensu and EventRank when using the monitoring-as-code concept. This concept is very valuable for ever scaling system that is constantly evolving over time. By being able to add code to a monitoring system rather that configuring a static file it is much easier to evolve the system without having to take it down for maintenance. This is something that is becoming more and more important as system keeps growing larger and larger and dynamic scaling become common. Another part of the implementation shows that it is possible to integrate an mathematical formula into a monitoring system without compromising the original functionality of actually reporting on error in the system. This is an important part of the thesis as the problem statement was to improve the situational awareness and therefor not eliminate already existing functions.

When looking back the thesis was a success as it was possible to use EventRank as an addition to the monitoring system and increased the situational awareness of the system. It can help detect and point out patterns of monitoring errors. This will help system administrators in every day operations and save time troubleshooting for hidden mistakes. The code was successfully implemented with the system and gave good feedback from the experiments performed to test the functionality. All in all this was a successfully feature and something that most monitoring systems should have as an addition to help get a better understanding.

Bibligraphy

[1] Altinn. Altinn Homepage. 2014. URL: https://www.altinn.no/en/?epslanguage=en

[2] Havey Water Operations, LLC(OR). Sensu homepage. 2014. URL: www.sensuapp.org

[3] Kyrre Begnum. Towards autonomic managment in system administration. June, 2008.

URL: http://www.iu.hio.no/~kyrre/phd_thesis_kyrre_begnum.pdf

[4] K. Begnum and M. Burgess. Priciple components and importance ranking of distributed anomalies. Jurnal: Machin learning Journal. June, 2005

[5] Openstack. Openstack homepage. 2014. URL: https://www.openstack.org/

[6] MySQL. MySQL homepage. 2014. URL: www.mysql.com

[7] OpenTSDB. OpenTSDB Homepage. 2014. URL: http://opentsdb.net

Chapter 7

Appendix

7.1 Appendix A: An example of Sensu check

```
{
"checks": {
  "event001": {
  "handler": "eventhandler",
  "command": "check_for_file.sh /tmp/test001",
  "interval": 20,
  "subscribers": [ "test" ],
  "refresh": 600
  }
  }
}
```

7.2 Appendix B: An example of sensu handler

```
{
"handlers": {
    "eventhandler": {
       "mutator" : "eventrank",
        "type": "pipe",
        "command": "/etc/sensu/handlers/erhandler.pl"
        }
    }
}
```

7.3 Appendix C: An example of handler who sends to mutator in sensu

```
{
"mutators": {
   "eventrank": {
      "type": "pipe",
       "command": "/etc/sensu/mutators/eventrank.pl"
    }
}
}
```

7.4 Appendix D: An example of how events where turned on and off

```
#!/bin/bash
### Interval 1 (39)
rm /tmp/test086
rm /tmp/test069
rm /tmp/test009
rm /tmp/test037
rm /tmp/test007
rm /tmp/test016
rm /tmp/test074
rm /tmp/test082
rm /tmp/test075
rm /tmp/test093
rm /tmp/test065
rm /tmp/test097
rm /tmp/test041
rm /tmp/test025
rm /tmp/test084
rm /tmp/test003
rm /tmp/test053
rm /tmp/test061
rm /tmp/test062
rm /tmp/test027
rm /tmp/test070
rm /tmp/test085
rm /tmp/test029
rm /tmp/test004
rm /tmp/test000
rm /tmp/test079
rm /tmp/test049
rm /tmp/test068
rm /tmp/test056
```

```
rm /tmp/test017
rm /tmp/test005
rm /tmp/test034
rm /tmp/test022
rm /tmp/test077
rm /tmp/test010
rm /tmp/test081
rm /tmp/test001
rm /tmp/test002
rm /tmp/test095
sleep 40
touch /tmp/test086
touch /tmp/test069
touch /tmp/test009
touch /tmp/test037
touch /tmp/test007
touch /tmp/test016
touch /tmp/test074
touch /tmp/test082
touch /tmp/test075
touch /tmp/test093
touch /tmp/test065
touch /tmp/test097
touch /tmp/test041
touch /tmp/test025
touch /tmp/test084
touch /tmp/test003
touch /tmp/test053
touch /tmp/test061
touch /tmp/test062
touch /tmp/test027
touch /tmp/test070
touch /tmp/test085
touch /tmp/test029
touch /tmp/test004
touch /tmp/test000
touch /tmp/test079
touch /tmp/test049
touch /tmp/test068
touch /tmp/test056
touch /tmp/test017
touch /tmp/test005
touch /tmp/test034
touch /tmp/test022
touch /tmp/test077
touch /tmp/test010
touch /tmp/test081
touch /tmp/test001
```

```
touch /tmp/test002
touch /tmp/test095

### Interval 2 (21)
....
```

7.5 Appendix E: An example of custom Sensu handler

```
#!/usr/bin/perl

use JSON::XS;
use Data::Dumper;

# event input from sensu
my $event = <STDIN>;

open(LOG,">>/var/log/sensu/erhandler.log");
print LOG time . " got line: $event\n";


my $hashref = decode_json $event;


my %client = %{$hashref->{"client"}};
my %check = %{$hashref->{"check"}};

print LOG Dumper(%client);
print LOG Dumper(%check);
```

7.6 Appendix F: How to create random occurences in the system

```
#!/usr/bin/perl
open(SCRIPT,">/etc/sensu/experiment_script_$ARGV[0].sh");
print SCRIPT "#!/bin/bash\n";

my $INTERVALS = $ARGV[1]; # the number of intervals to generate


for ( $i = 1; $i <= $INTERVALS; $i++){

    # Step 1 generate the number of events (max 50)
    my $eventnum = int(rand(20));

    # Step 2, pick the events to remove
    my %events;
    my $foundevents;
    while ( $foundevents < $eventnum ){

        my $candidate = int(rand(100));
        my $number = sprintf("%03s", $candidate);
        if ( not $events{"test$number"} ){
            $events{"test$number"} = 1;
            $foundevents++;
        }
    }

    # Step 3: Write the code into the script first remove,
    sleep, touch
    print SCRIPT "### Interval $i ($foundevents)\n";
    for my $event (keys %events){
        print SCRIPT "rm /tmp/$event\n";
    }
    print SCRIPT "sleep 40\n";
    for my $event (keys %events){
        print SCRIPT "touch /tmp/$event\n";
    }
    print SCRIPT "\n";

}
system("chmod +x /etc/sensu/experiment_script_$ARGV[0].sh");
```